

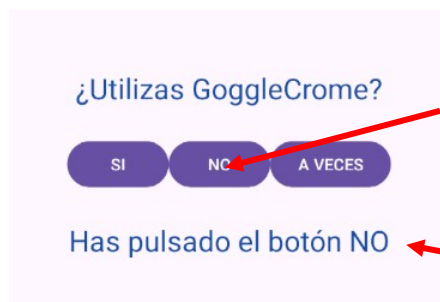
# EVENTOS

## 1. INTRODUCCION

- Hasta ahora hemos definido interfaces de usuario para poder crear pantallas en las aplicaciones, pero no podíamos interaccionar con ellas porque no hemos definido código para **responder a eventos**.
- De eso trataremos en este documento: veremos cómo responder ante distintas acciones que el usuario aplique sobre la pantalla y a leer valores que se introduzcan en las vistas.
- Las aplicaciones Android suelen estar **controladas por eventos**. A diferencia de otro tipo de aplicaciones, las aplicaciones controladas por eventos se inician y quedan a la espera de registrar dichos eventos, por ejemplo, registrar cuando se produce la pulsación de un botón por parte del usuario. Además, tanto el propio sistema operativo como una aplicación pueden iniciar eventos, pero los más significativos son los que inicia el propio usuario.

## 2. REPUESTA A LA PULSACION DE UN BOTON

- Cada vez que hacemos clic en uno de los botones de un layout, se produce un **evento onClick**, que llama al método con el mismo nombre (método `onClick()`).
- Los métodos que reciben eventos `onClick` deben ser de tipo público y nulos (**public void**). Y además deben aceptar como parámetro un objeto de tipo View, que referencia al elemento de la interfaz sobre el que hemos hecho clic.
- Existen diferentes formas de gestionar los eventos de un botón.  
Para probarlas, recuperaremos uno de nuestros proyectos.  
La primera modificación será añadir una TextView en el archivo de layout para visualizar el mensaje de respuesta a cada pulsación.



## 2.1. Gestionar el evento `onClick` mediante la propiedad `onClick` del elemento `Button` en el fichero `.xml`

- Es la opción más simple pero también la menos versátil.
- Actualmente se recomienda el uso de las opciones que veremos a continuación.
- Añadimos la propiedad **`android:onClick`** a cada botón:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:text="@string/txtSi"
    android:onClick="onClickBtnSi"/>
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:text="@string/txtNo"
    android:onClick="onClickBtnNo"/>
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:text="@string/txtAveces"
    android:onClick="onClickBtnAveces"/>
```

- El nombre indicado como valor del atributo “*android:onClick*” debe corresponderse con un método público y nulo. Así, en este ejemplo, debemos declarar los tres métodos públicos:

```
public void onClickSi(View v)
public void onClickNo(View v)
public void onClickAveces(View v)
```

- Para trabajar con los elementos de la interfaz desde el código, lo primero que hay que hacer es **recuperar el objeto** que los representa, mediante el método ***findViewById(identificador)***. Este método busca en el layout mostrado el elemento que tenga como identificador el que se haya pasado como parámetro. Y devuelve un objeto de tipo `View`, por lo que es necesario hacer una conversión explícita o cast a la clase correspondiente del objeto, para poder acceder a los métodos que sean necesarios:

```
TextView respuesta = (TextView) findViewById(R.id.respuesta);
```

(Actualmente ya no es obligatorio codificar esta operación de cast explícitamente)

- Y, por último, en cada método encargado de gestionar la pulsación de cada botón, se escribe el mensaje en el `TextView` de salida mediante el método ***setText()***.

```
respuesta.setText("Has hecho click en SI");
```

- Código resultante

```
TextView textViewRespuesta;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_linear);
    textViewRespuesta = findViewById(R.id.respuesta);
}

public void onClickBtnNo(View view) {
    textViewRespuesta.setText("Has pulsado el botón NO");
}

... y así con los demás ...
```

- En lugar de crear un método para cada botón, se puede crear **un único método**, común para todos los botones.

```
<Button
...
    android:onClick="onClickBtn"/>
```

- Y, por código, controlar qué botón fue el que se pulsó, mediante el método **getId()**.

```
public void onClickBtn(View view) {
    if (view.getId() == R.id.btnNo)
        textViewRespuesta.setText("Has pulsado el botón NO");
    else if (view.getId() == R.id.btnSi)
        textViewRespuesta.setText("Has pulsado el botón SI");
    else if (view.getId() == R.id.btnAveces)
        textViewRespuesta.setText("Has pulsado el botón A VECES");
}
```

## Comentario

Con este código es suficiente para que cada botón imprima su propio mensaje. Ahora bien, aunque es una opción sencilla no es la mejor opción.

Si definimos los eventos en el layout pero no los métodos en la Activity, el código compilará sin ningún problema y únicamente fallará en tiempo de ejecución cuando se percate de que el/los métodos no existen.

Para evitarlo, acostumbrémonos a **crear los métodos de forma automática** con el generador de código

Por ejemplo, si sustituimos

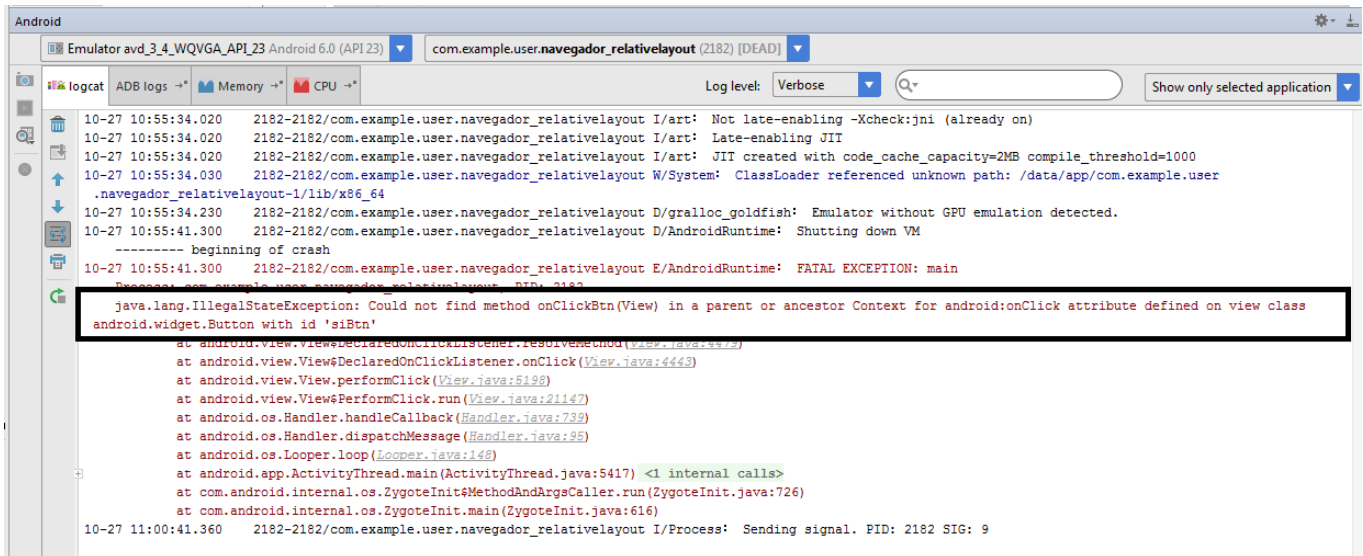
```
public void onClickBtn(View v) {...}
```

por

```
public void onClickbtn (View v) {...}
```

La aplicación es lanzada sin problema, pero rompe la ejecución cuando pulsamos uno de los botones:

Consultamos el archivo de log para ver el error que se ha producido:



## 2.2. Implementar un listener

- Cuando una aplicación se encuentra a la espera de que se produzca un determinado evento, se dice que está “a la escucha” de dicho evento. Así, el objeto que responde a un evento se denomina “escuchador” o “listener”.
- Un **listener** (escuchador de eventos) es una interfaz de la clase **View** que contiene un método que hay que sobrescribir.
- El SDK de Android incorpora interfaces de escuchador para diferentes eventos. Esto da lugar a que se puedan capturar diferentes eventos mediante listeners. P.ej. los eventos **onClick** (el que nos interesa en este momento referido a la pulsación de un botón), **onCreateContextMenu** (para crear un menú contextual), **onItemSelected** (para controlar cuando ha sido pulsado un elemento de una lista o spinner)...
- Interfaces de la clase View:

Documentación en <http://developer.android.com/reference/android/view/View.html>

nce > View.OnClickListener

### Summary

Public Methods	
abstract void	<code>onClick(View v)</code> Called when a view has been clicked.

### Public Methods

public abstract void	<code>onClick (View v)</code>	Added in API level 1
----------------------	-------------------------------	----------------------

Called when a view has been clicked.

**Parameters**

`v` The view that was clicked.

- Hay diferentes formas de implementar un escuchador (listener) en nuestro código. En cualquier caso, además de implementar dicho escuchador es necesario asignárselo a la vista correspondiente. Esto se hace mediante el método **`setOn____Listener()`**, en donde la línea representa el nombre del evento en cuestión, en este caso, **`setOnClickListener()`**

void	<code>setOnClickListener(View.OnClickListener l)</code> Register a callback to be invoked when this view is clicked.
------	---

### Método 1

Crear una clase que implemente la interfaz (**`OnClickListener`**, en este caso) y que se usará para crear a su vez el objeto escuchador propiamente dicho.

Este método no resulta adecuado porque la clase en sí no tiene más interés que el poder crear el objeto escuchador, es decir, no tiene ninguna otra finalidad (***pero puede servirnos para entender mejor los otros métodos***)

Ejemplo:

```
//creación de clase interna para implementar el escuchador
private class Aux implements View.OnClickListener{
    @Override
    public void onClick(View v) {
        //gestionar pulsación del btn1
        respuesta.setText("Botón 1 pulsado");
    }
}

//crear objeto de la clase auxiliar y asignarlo al botón 1
Aux escuchador1 = new Aux();
btn1.setOnClickListener(escuchador1);
```

## Método 2

Crear el objeto escuchador mediante una **clase anónima**.

Ejemplo:

```
//crear el escuchador med una clase anónima
private View.OnClickListener escuchador2 = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //gestionar pulsación del btn2
        respuesta.setText("Botón 2 pulsado");
    }
};

//asignar escuchador al btn2
btn2.setOnClickListener(escuchador2);
```

## Método 3

Crear el objeto escuchador y asignárselo a la vista en un solo paso.

Ejemplo:

```
//crear y asignar escuchador al btn3 en un solo paso
btn3.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        //gestionar pulsación del btn3
        respuesta.setText("Botón 3 pulsado");
    }
});
```

Comentario:

Al método **setOnClickListener()** se le pasa como parámetro la creación de una clase anónima que implementa una interfaz (**OnClickListener**), para lo cual hay que sobrescribir el único método que contiene dicha interfaz: **onClick()**.

## Método 4

Implementar la interfaz desde la propia clase **MainActivity**.

Ejemplo:

```
public class MainActivity extends AppCompatActivity
    implements View.OnClickListener {...}
```

```
//implementar metodo abstracto onClick
@Override
public void onClick(View v) {
    //gestionar pulsación del btn4
    respuesta.setText("Botón 4 pulsado");
}

//asignar escuchador al btn4 (dentro de onCreate)
btn4.setOnClickListener(this);
```

### 3. RESPUESTA A LA PULSACION DE OTROS TIPOS DE BOTON

Para probarlo, recuperaremos el proyecto **Componentes**.

#### 3.1. ToggleButton

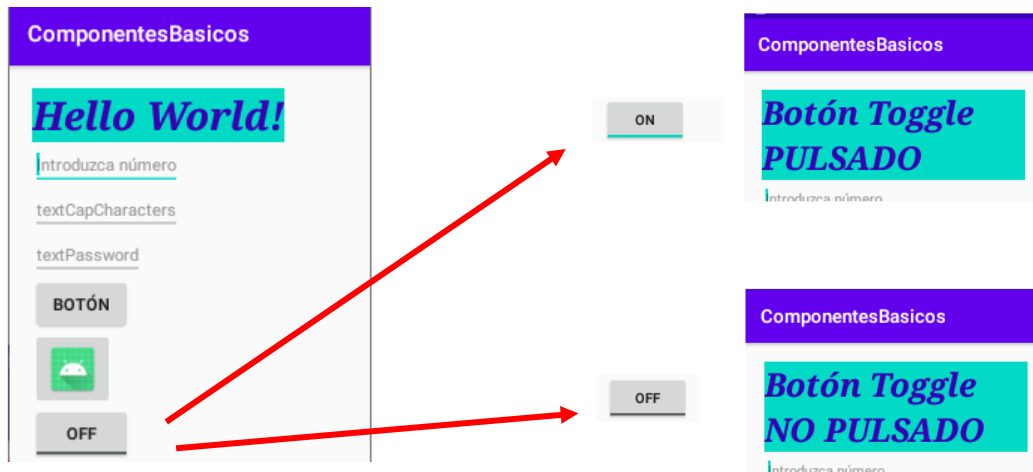
- Como son botones, también disponen de la propiedad **android:onClick**.
- Como tienen dos estados, suele ser de utilidad conocer en qué estado ha quedado el botón tras ser pulsado, para lo que podemos utilizar su método **isChecked()**, que devuelve un valor booleano.
- **Ejemplo de código:**

Archivo de layout

```
<ToggleButton
    android:id="@+id/btnToggle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOff="Off"
    android:textOn="On"
    android:onClick="onClickToggle"/>
```

Actividad principal

```
public void onClickToggle(View view) {
    if(btnToggle.isChecked()){
        txtRespuesta.setText("Botón Toggle PULSADO");
    }else{
        txtRespuesta.setText("Botón Toggle NO PULSADO");
    }
}
```



También se puede gestionar el evento mediante un **listener con clase anónima**, y eliminando la propiedad **android:onClick** del archivo .xml:

```
//Listener con clase anónima
btnToggle.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if(btnToggle.isChecked())
            txtRespuesta.setText("Botón Toggle PULSADO");
        else
            txtRespuesta.setText("Botón Toggle NO PULSADO");
    }
});
```

### 3.2. ImageButton

- Funciona igual que un botón normal.
- También funcionaría lo mismo si el botón se compone de texto más imagen.
- **Ejemplo de código:**

```
//Listener del ImageButton
imgBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        txtRespuesta.setText("Imagen pulsada");
    }
});
```

### 3.3. CheckBox

- Como ya se comentó en un documento anterior, este componente es similar al componente *ToggleButton*.




- Se puede comprobar su estado mediante el método **isChecked()**, que devuelve un valor booleano.
- Se puede establecer un estado concreto mediante el método **setChecked(estado)**.
- También puede ser interesante el método **toggle()**, que cambia el estado que tenga la checkbox en ese momento.

- **Evento onClick:**

- Para determinar el estado de una casilla de verificación podemos gestionar su evento **onClick**.
- Esto puede hacerse igual que con los botones: mediante la propiedad **android:onClick** o mediante listeners.
- Ejemplo con onClick:

```
<CheckBox
    android:id="@+id/chkWindows"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Windows"
    android:textColor="@color/colorAccent"
    android:onClick="onClickWindows"/>

public void onClickWindows(View view) {
    if (chkWindows.isChecked()){
        txtRespuesta.setText("Elegiste Windows");
    }
}
```




- Ejemplo con listener:

```
chkWindows.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (chkWindows.isChecked()){
            txtRespuesta.setText("Elegiste Windows");
        }
    }
});
```

También podemos hacer la consulta sobre el parámetro *view*, pero hay que hacer un cast a la clase **CheckBox**, ya que para un objeto **View** no existe el método **isChecked()**

```
chkLinux.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if ( ((CheckBox) view).isChecked() ){
            txtRespuesta.setText("Elegiste Linux");
        }
    }
});
```



- **Evento onCheckedChanged :**

- Informa de que ha cambiado el estado de la casilla de verificación.
- Ejemplo (con la segunda casilla de verificación):

```
chkWindows.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton compoundButton, boolean b) {
        if(b){
            txtRespuesta.setText("Estado ON");
        }else {
            txtRespuesta.setText("Estado OFF");
        }
    }
});
```

### 3.4. RadioButton

- Al igual que con checkbox, se pueden gestionar los eventos **onClick()** y **onCheckedChanged()**.
- También son métodos válidos **isChecked()**, **setChecked(boolean)** y **toggle()**
- Otro método útil de la clase RadioGroup es el método **getCheckedRadioButtonId()** que, como se puede extraer de su nombre, retorna el id del radiobutton seleccionado.
- Ejemplo con listener de clase anónima:

```
rbD.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        txtRespuesta.setText("Elegiste diurno");
    }
});
```

- Ejemplo de uso de evento onCheckedChanged

```
rg.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(RadioGroup radioGroup, int i) {
        if(i==R.id.rbDiurno){
            txtRespuesta.setText("Elegiste diurno");
        }else if(i==R.id.rbNocturno){
            txtRespuesta.setText("Elegiste noturno");
        }
    }
});
```

## 4. CAPTURA DE CONTENIDO DE EDITTEXT

- La captura del contenido de un EditText normalmente está asociada a un evento (por ejemplo, a la pulsación de un botón).
- Un **EditText** es el objeto más simple que podemos emplear en Android para leer entradas de texto.

- Hereda de la clase **TextView**.
- Para manipular un objeto de tipo EditText desde Java hay que utilizar el método **getText()**, el cual no devuelve directamente un String, sino un objeto Editable.

Editable	getText() Devuelve el texto que muestra TextView.
----------	--

- Sobre el objeto de tipo Editable emplearemos el método **toString()** para convertirlo en un dato de este tipo (es decir, para convertirlo en un string).
- Ejemplo:  

```
String numero = etNumero.getText().toString();
```